Software Quality Journal (2023) 31:1527–1559
https://doi.org/10.1007/s11219-023-09650-4

RESEARCH

Lessons learned from replicating a study on information-retrieval-based test case prioritization

Nasir Mehmood Minhas[1,2] · Mohsin Irshad[3] · Kai Petersen[1,4] · Jürgen Börstler[1]

Accepted: 9 September 2023 / Published online: 16 October 2023

Abstract
Replication studies help solidify and extend knowledge by evaluating previous studies' findings. Software engineering literature showed that too few replications are conducted focusing on software artifacts without the involvement of humans. This study aims to replicate an artifact-based study on software testing to address the gap related to replications. In this investigation, we focus on (i) providing a step-by-step guide of the replication, reflecting on challenges when replicating artifact-based testing research and (ii) evaluating the replicated study concerning the validity and robustness of the findings. We replicate a test case prioritization technique proposed by Kwon et al. We replicated the original study using six software programs, four from the original study and two additional software programs. We automated the steps of the original study using a Jupyter notebook to support future replications. Various general factors facilitating replications are identified, such as (1) the importance of documentation; (2) the need for assistance from the original authors; (3) issues in the maintenance of open-source repositories (e.g., concerning needed software dependencies, versioning); and (4) availability of scripts. We also noted observations specific to the study and its context, such as insights from using different mutation tools and strategies for mutant generation. We conclude that the study by Kwon et al. is partially replicable for small software programs and could be automated to facilitate software practitioners, given the availability of required information. However, it is hard to implement the technique for large software programs with the current guidelines. Based on lessons learned, we suggest that the authors of original studies need to publish their data and experimental setup to support the external replications.

Keywords Replication · Regression testing · Technique · Test case prioritization · Information retrieval · SIR

1 Introduction

Replications help evaluate results, limitations, and validity of studies in different contexts (Shull et al., 2008). They also help establish or expand the boundaries of a theory (Da Silva et al., 2014; Shull et al., 2008).

Extended author information available on the last page of the article

During the previous four decades, software engineering researchers have built new knowledge and proposed new solutions, but many of these lack consolidation (Juristo & Gómez, 2012; Krein & Knutson, 2010). Replication studies can help establish the solutions and expand the knowledge (Shepperd et al., 2018). Software engineering researchers have been working on replication studies since the 1990s. Still, the number of replicated studies is small in software engineering (Da Silva et al., 2014; de Magalhães et al., 2015), most software engineering replication studies are conducted for experiments involving human participants, and comparatively few replications exist for artifact-based experiments (Da Silva et al., 2014). In the artifact-based software engineering experiments, the majority of the authors use the artifacts from the software infrastructure repository (SIR) (Yoo & Harman, 2012). Do et al. (2005) introduced SIR in 2005 to facilitate experimentation and evaluation of testing techniques (mainly regression testing techniques) and to promote replication of experiments and aggregation of findings.

In this study, we focus on the possibility of replicating regression testing techniques. Regression testing is a complex and costly activity. Practitioners face various challenges while performing regression testing and are interested in techniques that can fulfill their goals (Engström & Runeson, 2010; Minhas et al., 2020). Researchers have been proposing different techniques to support regression testing practice, and some of them are evaluating their techniques in an industry context. However, adopting these techniques in practice is challenging because, in most cases, the results are inaccessible for practitioners (Ali et al., 2019). Moreover, most regression testing techniques proposed in research have been evaluated using open-source data sets (Yoo & Harman, 2012). Adopting these techniques in practice is challenging because practitioners do not know the context these techniques can fit. Replications of existing solutions for regression testing can be helpful in this regard, provided data and automation scripts are available for future replications.

Attempts have been made to replicate regression testing techniques. The majority of these replications are done by the same group of authors who originally proposed the techniques (Do & Rothermel, 2006; Do et al., 2004, 2010). There is a need for conducting more independent replications in software engineering (Do et al., 2005). However, evidence of independent replications in regression testing is low (Da Silva et al., 2014).

Overall, we would highlight the following research gaps concerning replications:

- *Gap 1: A small portion of studies are replications, especially lack of replications in specific subject areas:* Software engineering is lacking in replication studies (Da Silva et al., 2014). In particular, software testing as a subject area has been highlighted as an area lacking replication studies (Da Silva et al., 2014). According to Da Silva et al. (2014), the majority of replication studies focus on software construction and software requirements.
- *Gap 2: Lack of replication guidelines:* Software engineering research lacks standardized concepts, terminologies, and guidelines (de Magalhães et al., 2015; Krein & Knutson, 2010). There is a need to work on the guidelines and methodologies to support replicating the studies (de Magalhães et al., 2015).
- *Gap 3: Lack of studies on artifact-based investigations:* Only a few replicated studies focused on artifact-based investigations (Da Silva et al., 2014). That is, the majority of studies focused on experiments and case studies involving human subjects. Artifact-based replications are of particular interest as they require building and running scripts for data collection (e.g., solution implementation and logging) and simultaneously compiling and running the software systems, which are the subject of study.

Considering the gaps stated above, we formulate the following research aim:

Aim: *To replicate an artifact-based study in the area of software testing, with a focus on reflecting on the replication process and the ability to replicate the findings of the original study.*

To achieve our research aim, we present the results of our replication experiment in which we evaluated an IR-based test case prioritization technique proposed by Kwon et al. (2014). The authors introduced a linear regression model to prioritize the test cases targeting infrequently tested code. The inputs for the model are calculated using term frequency (TF), inverse document frequency (IDF), and code coverage information (Kwon et al., 2014). TF and IDF are the weighing schemes used with information retrieval methods (Roelleke, 2013). The original study's authors used open-source data sets (including SIR artifacts) to evaluate the proposed technique. We attempted to evaluate the technique using six software programs to see if the replication confirms the original study's findings. We used all four software programs from the original study and two new cases to test the technique's applicability to different software programs.

In the pursuit of our research aim, we achieved the following two objectives:

1. *Objective 1: Studying the extent to which the technique is replicable.* Studying the extent to which the technique is replicable and documenting the detail of all steps will help draw valuable lessons. Hence, contributing with guidance for future artifact-based replications (Gap 2, Gap 3).
2. *Objective 2: Evaluating the results of the original study* (Kwon et al., 2014). Evaluating the results through replication provides an assessment of the validity and robustness of the original study's results. Overall, we contribute to the generally limited number of replication studies in general and replication studies focused on software testing in particular (Gap 1).

The organization of the rest of the paper is as follows: Sect. 2 briefly introduces the concepts relevant to this study. Section 3 briefly discusses some replications carried out for test case prioritization techniques. Along with the research questions and summary of the concepts used in the original study, Sect. 4 describes the methodologies we have used to select the original study and conduct the replication. Section 5 presents the findings of this study, and Sect. 6 provides a discussion on the findings of the replication study. Threats to the validity of the replication experiment are discussed in Sect. 7, and Sect. 8 concludes the study.

2 Background

This section provides a discussion on the topics related to our investigation.

2.1 Regression testing

Regression testing is a retest activity to ensure that system changes do not negatively affect other parts of the system and that the unchanged parts are still working as before a change (Minhas et al., 2020; Yoo & Harman, 2012). It is an essential but expensive and challenging testing activity (Engström & Runeson, 2010). Various authors have highlighted that testing consumes 50% of the project cost, and regression testing consumes 80% of the total testing cost (Engström et al., 2010; Engström & Runeson, 2010; Harrold & Orso, 2008; Kazmi et al., 2017). Research reports that regression testing may consume more than 33% of the cumulative software cost (Khatibsyarbini et al., 2018). Regression testing aims to validate that modifications have not affected the previously working code (Do et al., 2010; Minhas et al., 2020).

Systems and Software Engineering–Vocabulary (ISO/IEC/IEEE, 2017), defines regression testing as follows:

1. "Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements."
2. "Testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects."

For larger systems, executing regression test suites in full is expensive (Minhas et al., 2020). One of the suggested solutions is test case prioritization to cope with this. It helps to prioritize and run the critical test cases early in the regression testing process. The goal of test case prioritization is to increase the test suite's rate of fault detection (Elbaum et al., 2002).

A reasonable number of systematic literature reviews and mapping studies on various aspects of regression testing provides evidence that regression testing is a well-researched area (Ali et al., 2019; Bajaj & Sangwan, 2019; Catal, 2012; Catal & Mishra, 2013; Dahiya & Solanki, 2018; Engström et al., 2010; Felderer & Fourneret, 2015; Kazmi et al., 2017; Khatibsyarbini et al., 2018; Lima & Vergilio, 2020; Qiu et al., 2014; Rosero et al., 2016; Singh et al., 2012; Yoo & Harman, 2012; Zarrad, 2015). Despite many regression testing techniques proposed in the literature, adopting these techniques in the industry is low (Ekelund & Engström, 2015; Engström & Runeson, 2010; Rainer et al., 2005; Rainer & Beecham, 2008). The reasons are that the results of these techniques are not accessible for practitioners due to the discrepancies in terminology between industry and academia (Ali et al., 2019; Engström & Runeson, 2010; Minhas et al., 2017). There is a lack of mechanisms to guide the practitioners in translating, analyzing, and comparing the regression testing techniques. Furthermore, various authors use controlled experiments for their empirical investigations. In most cases, it is hard to assess that these experiments are repeatable and could fit in an industrial setting (Ali et al., 2019). Replication of empirical studies could lead us to the desired solution, as it can help to confirm the validity and adaptability of these experiments (Shull et al., 2008).

2.2 Replication

Replication is a means to validate experimental results and examine if the results are reproducible. It can also help to see if the results were produced by chance or if the results are the outcome of any feigned act (Juristo & Gómez, 2012). An effectively conducted replication study helps in solidifying and extending knowledge. In principle, replication provides a way forward to create, evolve, break, and replace theoretical paradigms (Krein & Knutson, 2010; Shull et al., 2008). Replication could be of two types (1) internal replication—a replication study carried out by the authors of the original study themselves and (2) external replication—a replication study carried out by researchers other than the authors of the original study (Shepperd et al., 2018; Krein & Knutson, 2010).

In software engineering research, internal replications are much higher than external replications (Bezerra et al., 2015; Da Silva et al., 2014). Da Silva et al. (2014) reported in their mapping study that out of 133 included replication studies, 55% of the studies are internal replications, 30% are external replications, and 15% are a mix of internal and external. Furthermore, the results of 82% of the internal replications are confirmatory, and the results of 26% of external replications conform to the original studies (Da Silva et al., 2014). From the empirical software engineering perspective, Shull et al. (2008) classify replications as exact and conceptual replication. In an exact replication, the replicators closely follow the procedures of the original experiment. In a conceptual replication, the research questions of the original study are evaluated using a different experimental setup. Concerning exact replication, if the replicators keep the conditions in the replication experiment the same as the actual experiment, it would be categorized as exact dependent replication. If replicators deliberately change the underlying conditions of the original experiment, it would be referred to as exact independent replication. Exact dependent and exact independent replications could respectively be mapped to strict and differentiated replications. A strict replication compels the researchers to replicate a prior study as precisely as possible. In contrast, in a differentiated replication, researchers could intentionally alter the aspects of a previous study to test the limits of the study's conclusions. In most cases, strict replication is used for both internal and external replications (Krein & Knutson, 2010).

2.3 Information retrieval

IR-based techniques are used to retrieve the user's information needs from an unstructured document collection. The information needs are represented as queries (Fang et al., 2004; Yadla et al., 2005). An information retrieval (IR) system is categorized by its retrieval model because its effectiveness and utility are based on the underlying retrieval model (Amati, 2009). Therefore, a retrieval model is the core component of any IR system.

Amati (2009) defines the information retrieval model as follows:

"A model of information retrieval (IR) selects and ranks the relevant documents with respect to a user's query. The texts of the documents and the queries are represented in the same way, so that document selection and ranking can be formalized by a matching function that returns a retrieval status value (RSV) for each document in the collection. Most of the IR systems represent document contents by a set of descriptors, called terms, belonging to a vocabulary V."

Some of the retrieval models are the vector space model (VSM), probabilistic relevance framework (PRF), binary independence retrieval (BIR), best match version 25 (BM 25), and language modeling (LM). VSM uses TF-IDF (term frequency and inverse document frequency) as a weighing scheme, and it is among the popular models in information retrieval systems (Roelleke, 2013). For example, Pannu et al. (2014) compared different information retrieval models and concluded that VSM was the most effective compared to boolean and probabilistic models.

Since the technique (Kwon et al., 2014) we are replicating in this study uses the concepts of TF-IDF weighing scheme, we briefly present TF and IDF.

Term frequency (TF) and inverse document frequency (IDF) are statistics that indicate the significance of each word in the document or query. TF represents how many times a word appears in the document or query. IDF is an inverse of document frequency (DF). The DF of a word indicates the number of documents in the collection containing the word. Therefore, a high IDF score of any word means that the word is relatively unique, and it appeared in fewer documents (Fang et al., 2004).

3 Related work

In this section, we have included only studies replicating the test case prioritization techniques. Most of the replication studies on test case prioritization were conducted by the same group of authors, who primarily re-validated/extended the results of their previously conducted experiments (see Do et al., 2004, 2010; Do & Rothermel, 2006). Below we discuss studies that are closely related to our topic (i.e., test case prioritization).

Do et al. (2004) conducted a replication study to test the effectiveness of the test case prioritization techniques originally proposed for C programs on different Java programs using the JUnit testing framework. The authors' objective was to test whether the techniques proposed for C programs could be generalized to other programming and testing paradigms. The authors who conducted the replication study were part of the original studies, so by definition, it could be referred to as an internal replication. However, concerning the implementation perspective, the replication study would be considered differentiated replication.

Do and Rothermel (2006) conducted an internal replication study to replicate one of their studies on test case prioritization. The original study used hand-seeded faults. In the replication study, the authors conducted two experiments. In the first experiment, the authors considered mutation faults. The goal was to assess whether prioritization results obtained from hand-seeded faults differ from those obtained from mutation faults. The authors used the same software programs and versions used in the original study. They also replicated the experimental design according to the original study. To further strengthen the findings, later in the second experiment, the authors replicated the first experiment with two additional Java programs with different types of test suites.

Ouriques et al. (2018) conducted an internal replication study of their own experiment concerning the test case prioritization techniques. The authors experimented with software programs closer to the industrial context in the original study. The objective of the replication study was to repeat the conditions evaluated in the original study but with more techniques and industrial systems as objects of study. Although the authors worked with the test case prioritization techniques, they clearly stated that the methods examined in their research use a straightforward operation of adding one test case at a time in the prioritized set. They do not use any data from the test case execution history; hence, regression test prioritization is not in the scope of their study.

Hasnain et al. (2019) conducted a replication study to investigate the regression analysis for classification in test case prioritization. The authors' objective was to replicate the original study to confirm whether or not the regression model used in the original study accurately produced the same results as the replicated study. Along with the program and data set used in the original study, the authors also used an additional open-source Java-based program to extend the original study's findings. It is an external replication study as all authors of the replication study differ from the original study. The authors of the replicated study validated the original study's results on an additional dataset other than the one used in the original study, and the replication is not strict.

In the above discussion of related studies, we learned that most replication studies conducted for test case prioritization are primarily internal replications. We could only find a single external replication study (Hasnain et al., 2019). The authors of this study replicated a classification-based test case prioritization using regression analysis. Our study is similar to this study based on the following factors, (1) our study is an external replication and (2) we also use four software artifacts from the original study and two additional artifacts. In many ways, our study is unique; for example, (1) we are replicating a technique that focuses on less tested code, whereas Husnain et al. replicated a technique that is based on fault classification and non-faulty modules, (2) we have provided a step by step guide to support future replications, and (3) we provide automated scripts to execute the complete replication study.

4 Methodology

We followed the guideline proposal for reporting the replication steps provided by Carver (2010). It suggests reporting the following for a replication study:

1. Information about the original study (Sect. 4.2)
2. Information about the replication (Sect. 4.3.2)
3. Comparison of results to the original study (Sect. 5.2)
4. Drawing conclusions across studies (Sect. 8)

4.1 Research questions

In the presence of the constraint regarding experimental setup and data, we have to rely on the information presented in the original study (see Sect. 4.2). We decided not to tweak the original study's approach, followed the steps proposed by the authors, and executed the technique on one of the artifacts used by the authors. The differential aspects of the replication experiment are the mutants and the automation of the major steps of the technique. According to the classification provided by Shull et al. (2008), our work can be classified as *exact independent replication* of the test case prioritization technique presented by Kwon et al. (2014).

To achieve the objectives of the study, we formulated the following two research questions:

RQ1. To what degree is the study replication feasible, given the information provided?

- RQ1.1 To what degree is the study replicable with the software programs used in the original study?
- RQ1.2 What is the possibility of replicating the study with the additional software programs?

The answer to RQ1 corresponds to Objective 1. While answering RQ1, the motive was to see the possibility of replicating the technique presented in the original study using different software programs.

RQ2. Does the replication confirm the findings of the original study?

The answer to RQ2 corresponds to Objective 2. The motive of RQ2 was to see if the replication results conform to the original study's findings. To ensure that there should be no conscious deviation from the basic technique, we followed the steps and used the tools mentioned in the original study. Finally, we evaluated the replication results using the average percentage of fault detection (APFD) as suggested by the original study's authors.

4.2 Information about the original study

4.2.1 Selection of target study

Selecting a target study for replication is difficult and often prone to bias (Pittelkow et al., 2021). For example, clinical psychology research reports that authors tend to choose targets that are easy to set up and execute (Pittelkow et al., 2021). The selection of a target must be purpose-based, either by following systematic criteria (see, e.g., Pittelkow et al., 2021) or by other justifiable reasons.

Our selection of study to be replicated is based on the needs identified in our previous studies (Ali et al., 2019; Minhas et al., 2017, 2020) and reported use of SIR systems for replications (Singh et al., 2012; Yoo & Harman, 2012). In the mentioned studies, we found that practitioners are looking for regression testing techniques that can fulfill their goals (e.g., controlling fault slippage and increasing fault detection ratio) (Minhas et al., 2017, 2020). In search of techniques that can fulfill practitioners' goals, Ali et al. (2019) conducted a systematic literature review. However, no single technique is reported in Ali et al. (2019) that works for controlling fault slippage. Based on practitioners' needs, we defined the following constraints for the target study.

- A test case prioritization technique that can control fault slippage and increase fault detection ratio.
- A technique that has been evaluated using SIR artifacts.

The reasons for setting these constraints are that during our investigations (Minhas et al., 2020), we identified that test case prioritization is among the primary challenges for practitioners. They are interested in finding techniques that can overcome their challenges and help them follow their goals (see also Minhas et al., 2017). Increasing a test suite's rate of fault detection is a common goal of regression test prioritization techniques (Lima & Vergilio, 2020; Pan et al., 2022), whereas controlling fault slippage is among the goals of the practitioners (Minhas et al., 2017, 2020).

Our next constraint was to select a study where authors used the SIR systems to evaluate their technique because SIR is the most used repository for regression testing techniques (Yoo & Harman, 2012), and SIR aims to facilitate replication (Do et al., 2005). Singh et al. (2012) reported that out of 65 papers selected for their systematic reviews on regression test prioritization, 50% are using SIR systems. Yoo and Harman (2012) also reported that most of the authors evaluate their techniques using SIR artifacts. They highlight that the use of SIR systems allows replication studies.

The final constraint was to select a target study that uses IR methods for the prioritization technique. Recent studies report that test case prioritization techniques based on IR concepts could perform better than the traditional coverage-based regression test prioritization techniques (Peng et al., 2020; Saha et al., 2015).

We searched Google Scholar with the keywords *"regression testing," "test case prioritization," "information retrieval," "IR," "software infrastructure repository," "SIR"*. Our searches returned 340 papers. After scanning the titles and abstracts, we learned that no single technique explicitly states controlling fault slippage as its goal. However, the technique presented in Kwon et al. (2014) focused on less tested code, and the goal was to increase the fault detection rate of coverage-based techniques using IR methods. Ignored or less tested code could be among the causes of fault slippage. Therefore we considered the technique by Kwon et al. (2014) for further evaluation. We evaluated this technique using the rigor criteria as suggested by Ivarsson and Gorschek (2011). The authors suggest evaluating the rigor of empirical studies based on context, design, and validity threats.

After considering all factors mentioned above and applying the rigor criteria, the study presented in (Kwon et al., 2014) was used as a target for replication.

4.2.2 Describing the original study

Kwon et al. (2014) intended to improve the effectiveness of test case prioritization by focusing on infrequently tested code. They argued that test case prioritization techniques based on code coverage might lack fault detection capability. They suggested that using the IR-based technique could help overcome the limitation of coverage-based test case prioritization techniques. Considering the frequency at which code elements have been tested, the technique uses a linear regression model to determine the fault detection capability of the test cases. Similarity score and code coverage information of test cases are used as input for the linear regression model. Kwon et al. (2014) stated that the technique they proposed is the first of its type that considers less tested code and uses TF-IDF in IR for test case prioritization. The authors claimed that their approach is also first in using linear regression to weigh the significance of each feature regarding fault-finding. They divided the process into three phases, i.e., validation, training, and testing, and suggested using the previous fault detection history or mutation faults as validation and training data.

Kwon et al. (2014) suggested the following steps to implement the proposed technique:

1. *Coverage of each test case*
2. *Set IDF threshold with validation data (previous or mutation faults)*
3. *Calculate TF/IDF scores of each test case*
4. *Use coverage and sum of TF/IDF of a test case as predictor values in the training data*
5. *Use previous/mutation faults as response values in the training data*
6. *Estimate the regression coefficients (weight of each feature) with the training data*
7. *Assign predictor values (coverage and TF/IDF scores) to the model to decide the test schedule*
8. *Run the scheduled test cases*

To evaluate the proposed test case prioritization technique (IRCOV), Kwon et al. (2014) used four open-source Java programs XML-Security (XSE), Commons-CLI (CCL), Commons-Collections (CCN), and Joda-Time (JOD). Kwon et al. (2014) highlighted that the fault information of the software programs was not sufficiently available, and they were unable to evaluate their approach using available information. Therefore, the authors simulated the faults using mutation. They used the mutation framework MAJOR (Just, 2014; Just et al., 2011) to generate the mutants. To reduce the researcher's bias and achieve reliable results, they applied ten-fold validation and divided the mutation faults into ten subsets, and assigned each subset to training, validation, and test data.

4.2.3 Concepts used in the original study

The original study (Kwon et al., 2014) makes use of IR concepts. It views a "document" as a test case, "words" as elements covered (e.g., branches, lines, and methods), and "query" as coverage elements in the updated files. TF and IDF scores of the covered elements determine their significance to a test case. The number of times a test case exercises a code element is counted as a TF value. The document frequency (DF) represents the number of test cases exercising an element. IDF is used to find the unique code elements as it is the inverse of DF.

Since the focus of the proposed technique was on less-tested code, the IDF score has more significance and is required to minimize the impact of TF. To minimize the impact of TF score on the test case prioritization, they used Boolean values for TF (i.e., $TF = 1$ if a test case covers the code element, $TF = 0$ otherwise). The IDF threshold is used to assign an IDF score to a code element. Kwon et al. (2014) define the IDF threshold as:

"The maximum number of test cases considered when assigning an IDF score to a code element."

The IDF threshold is determined by the validation data consisting of faults and related test cases from the previous test execution history or mutation faults.

Finally, the authors used the similarity score between a test case (document) and the changed element (query) to indicate the test cases related to modifications. The similarity score is measured using the sum of TF-IDF scores of common elements in the query.

4.2.4 Key findings of the original study

Using four open-source Java programs, the authors compared their technique with random ordering and standard code-coverage-based methods (i.e., line, branch, and method coverage). They measured the effectiveness using Average Parentage of Fault Detection *(APFD)*.

The authors concluded that their technique is more effective as it increased the fault detection rate by 4.7% compared to random ordering and traditional code coverage-based approaches.

4.3 Information about the replication

We first present contextual information, i.e., data availability (Sect. 4.3.1), and Sect. 4.3.2 presents how the replication steps were implemented.

4.3.1 Data availability

Kwon et al. (2014) did not provide details on the experimental setup and the raw data. Furthermore, they did not publish the automation scripts and data in an open-source repository. We contacted them and asked if they could share the experimental package with us. One of them informed us that they had lost the data and did not have any backups related to this study.

4.3.2 Replication steps

We aimed to make an exact replication of the original study, and therefore we followed the procedure strictly as presented in the original study (Kwon et al., 2014). The original study IRCOV was built using line, branch, and method coverage. Therefore, we also used the line, branch, and method coverage to replicate the IRCOV model. The sequence of events starts with generating mutants, then partitioning them into training, validation, and test data. Later, using validation data, IDF threshold is established, and using IDF threshold and training data, IDF scores are established. In the next step, TF and IDF scores are used to calculate similarity scores, whereas coverage is calculated using training data. Based on coverage data and similarity scores, regression coefficients are measured, and finally, using the regression model, the test schedule is decided. The detail of the steps followed in the replication experiment is shown in Fig. 1.

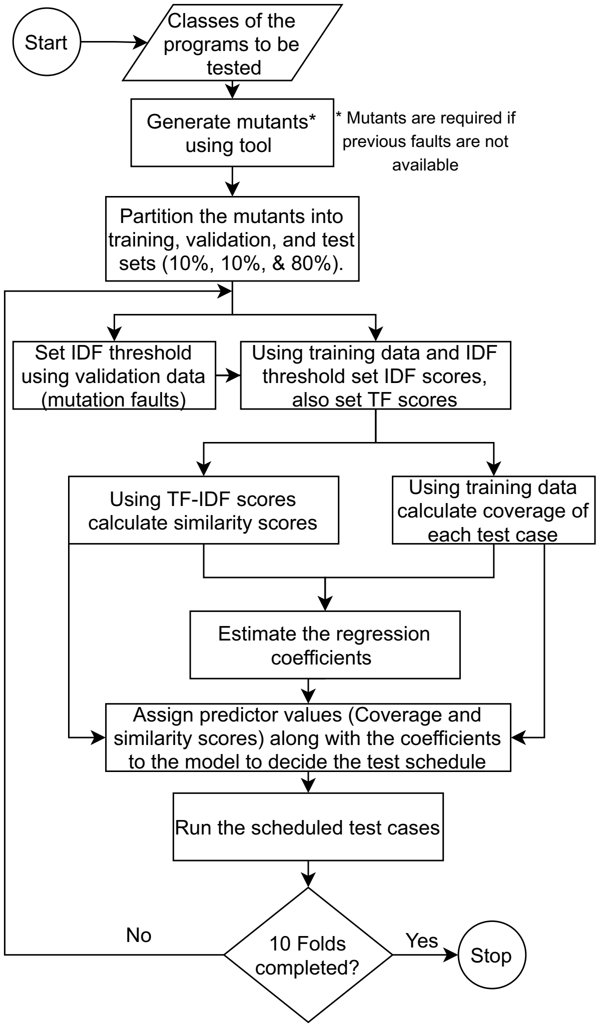

Start
Classes of the programs to be tested
Generate mutants* using tool
* Mutants are required if previous faults are not available
Partition the mutants into training, validation, and test sets (10%, 10%, & 80%).
Set IDF threshold using validation data (mutation faults)
Using training data and IDF threshold set IDF scores, also set TF scores
Using TF-IDF scores calculate similarity scores
Using training data calculate coverage of each test case
Estimate the regression coefficients
Assign predictor values (Coverage and similarity scores) along with the coefficients to the model to decide the test schedule
Run the scheduled test cases
10 Folds completed?
No
Yes
Stop

Fig. 1 Steps followed to replicate the original study

Table 1 Software programs used in replication

Program	Version	LOC	Test classes	Used in Kwon et al. (2014)	Repository
Commons-CLI	1.1, 1.2	13210	23	Yes	SIR & GitHub
XML Security	2.2.3	21315	172	Yes	SIR & GitHub
Commons-Collections	v4.3	128907	490	Yes	GitHub
Joda-Time	v2.10.10	147025	240	Yes	GitHub
Commons-Email	fd6b6**c35a	83154	20	No	GitHub
Log4j	b956**8b969	169646	63	No	SIR & GitHub

Replication objects: We attempted to test the replication of the IRCOV with six software programs, using four software programs from the original study and two additional software programs. Table 1 presents the details of the software programs used in the replication of IRCOV. The software programs are Common CLI, XML Security, Commons-Collections (CCN), Joda-Time (JOD), Commons-Email, and Log4j. We were able to implement IRCOV with Commons-CLI, but due to various constraints discussed in Sect. 5, we failed to replicate IRCOV with XML Security, Commons-Collections (CCN), Joda-Time (JOD), Commons-Email, and Log4j.

Commons-CLI[1] is a library providing an API parsing of command-line arguments. XML-Security[2] for Java is a component library implementing XML signature and encryption standards. Commons-Collections[3] provides Java data structures. Joda-Time[4] provides a quality replacement for the Java date and time classes.

To see if the technique (IRCOV) is replicable with other software programs, we selected Commons-Email and Log4j. Commons-Email[5] is built on top of the JavaMail API, and it aims to provide an API for sending emails. We selected Commons-Email because it is commonly used in regression testing studies (see, e.g., Legunsen et al., 2016, 2017; Zhang, 2018). Kwon et al. (2014) used two software programs from the Commons library (i.e., Commons-CLI and Commons-Collection). It was interesting to see if IRCOV is replicable for more programs from the Commons library.

Log4j[6] is a Java-based logging utility. Log4j 2 was released in 2014 to overcome the limitations of its predecessor version, Log4j 1. We obtained the software programs from GitHub and used the test suites provided with the programs. We selected Log4j because it is a larger program compared to the ones used in the original study (see Table 1). The intention was to see if IRCOV could be scaled up for larger programs. Moreover, Log4j is a program frequently used in regression testing experiments (see, e.g., Chen et al., 2018; Chi et al., 2020; Yu et al., 2014).

[1] https://commons.apache.org/proper/commons-cli/
[2] http://santuario.apache.org/javaindex.html
[3] https://commons.apache.org/proper/Commons-Collections/
[4] https://www.joda.org/joda-time/
[5] https://commons.apache.org/proper/Commons-Email/
[6] https://logging.apache.org/log4j/2.x/

Mutant generation: The fault information of the software programs was not available, and therefore we used mutation faults instead. In the original study, Kwon et al. (2014) used mutation faults as well. For the mutation, we used MAJOR (a mutation tool) (Just, 2014; Just et al., 2011).

Partitioning mutants into training, validation, and test sets: As per the description in the original study, we split the mutants into training, validation, and test sets (10%, 10%, and 80%, respectively). To split the data, we used an online random generator.[7] We applied the ten-fold validation technique to ensure the reliability of the results and avoid any bias. To create ten folds of each data set (i.e., training, validation, and test sets), we wrote automation scripts (Minhas & Irshad, 2021).

IDF threshold: The purpose of setting up an IDF threshold is to ensure that prioritized test cases should detect faults in less-tested code elements. The IDF threshold is decided using validation data containing information on faults and test cases detecting the faults. To calculate the IDF threshold, the authors of the original study (Kwon et al., 2014) suggested using a ratio from 0.1 to 1.0 in Eq. (1).

$$IDF\ Threshold = no\ of\ test\ cases \times ratio \tag{1}$$

We trained the regression model with each threshold using validation data and selected the ratio that led to the minimum training error for the IDF threshold. Based on the minimum training error, Table 2 presents the chosen values for the IDF threshold of all ten folds of Commons-CLI. We assigned IDF values to only those code elements whose DF was not above the IDF threshold.

Calculating TF and IDF score: As suggested in the original study (Kwon et al., 2014), we use Boolean values for TF (i.e., $TF = 1$ if the test case covers the element, $TF = 0$ otherwise). The purpose of fixing the TF values as 0 or 1 was to ensure that only test cases would be prioritized that are targeting less tested code. The IDF score is more significant in this regard. As suggested in the original study (Kwon et al., 2014), we used Eq. (2) to calculate the IDF score.

$$IDF = 1 + log\left(\frac{\#\ of\ test\ cases}{\#\ of\ test\ cases\ covering\ the\ element}\right) \tag{2}$$

Similarity score: The similarity score directly contributes to the IRCOV model, in the regression model (see Eq. 4), x_2 refers to the similarity score of each test case. We have calculated the similarity scores using Eq. (3), as suggested in Kwon et al. (2014). In Eq. (3), e refers to covered elements, q refers to changed elements, and t refers to a test case. Whereas tf and idf are term frequency and inverse document frequency.

$$Similarity\ Score(t, q) = \sum_{e \in t \cap q} tf - idf_{e,t} \tag{3}$$

Since TF values are 1 or 0 (i.e., if a test case excises a code element, then TF is 1; otherwise, it is 0), practically, similarity scores are the sum of IDF scores of the elements covered by a particular test case.

[7] https://approsto.com/random-line-picker/

Table 2 Simulation parameters for Commons-CLI. (MC, method coverage; LC, line coverage; BC, branch coverage)

Fold name	Coverage type	Training error	IDF threshold	θ_0	θ_1	θ_2
Fold1	MC	1.0694	7	−0.3478	0.0187	0.1426
	LC	0.9770	2	0	0	0
	BC	0.8876	2	0	0	0
Fold2	MC	0.3195	5	−0.7976	0.0323	−0.1472
	LC	0.3533	6	−0.6084	0.0088	−0.1343
	BC	0.3567	5	−0.3386	0.0178	−0.2095
Fold3	MC	0.6411	6	−0.0286	0.0008	0.0796
	LC	0.6404	6	−0.0498	0.0004	0.0736
	BC	0.6405	6	−0.0380	0.0008	0.0736
Fold4	MC	0.4783	6	−0.0687	0.0097	0.1677
	LC	0.4551	6	−0.1086	0.0032	0.1365
	BC	0.4947	6	0.1240	0.0045	0.1683
Fold5	MC	0.1838	5	0.0309	0.0068	0.0558
	LC	0.1856	4	0.0859	0.0018	0.0612
	BC	0.1876	4	0.1406	0.0038	0.0516
Fold6	MC	0.2247	2	−0.5284	0.0194	0.3548
	LC	0.1795	2	−0.4869	0.0052	0.3470
	BC	0.1549	2	−0.3978	0.0119	0.3149
Fold7	MC	0.1382	10	−0.1479	0.0115	−0.0141
	LC	0.1364	10	−0.0833	0.0030	−0.0234
	BC	0.1390	10	0.0028	0.0065	−0.0235
Fold8	MC	0.2020	6	0.4389	−0.0024	0.0839
	LC	0.2046	6	0.3401	−0.0001	0.0715
	BC	0.2046	6	0.3286	−0.00001	0.0694
Fold9	MC	0.1490	6	0.1652	−0.0032	0.1473
	LC	0.1532	6	0.0540	−0.0002	0.1344
	BC	0.1517	6	0.0862	−0.0012	0.1434
Fold10	MC	0.0339	10	−0.1253	0.0017	0.0267
	LC	0.0339	10	−0.1127	0.0004	0.0261
	BC	0.0343	10	−0.0920	0.0007	0.0278

Coverage information: The coverage measure is also used in the regression model. In Eq. (4), x_1 refers to the coverage size of each test case. To measure code size (line of code) and coverage of each test case, we used JaCoCo[8] (the same tool was used in the original study (Kwon et al., 2014)).

IRCOV model: We used Eq. (4) for the linear regression model as suggested in the original study (Kwon et al., 2014).

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x2 \tag{4}$$

[8] https://www.eclemma.org/jacoco/

In Eq. (4), x_1 is the size of the coverage data for each test case, and x_2 refers to the similarity score of each test case. The value of y represents each test case's fault detection capability, which is proportional to the number of previous faults detected by the test case. In the regression model, three coefficients need to be calculated (i.e., θ_0, θ_1, & θ_2). Here θ_0 represents the intersect, whereas, to calculate θ_1 and θ_2 Kwon et al. (2014) suggested using Eq. (5), which uses y value and respective values of x_1 and x_2. Here y could be calculated using Eq. (6), whereas x_1 and x_2 respectively represent the size of coverage and similarity scores of each test case.

$$theta = (X^T X)^{-1} X^T \mathbf{y} \tag{5}$$

$$y = \sum_{n=1}^{n} \frac{f_i}{log(t_i) + 1} \tag{6}$$

Prioritization based on fault detection capability: After having the values of coefficients and variables of regression model (i.e., θ_0, θ_1, θ_2, x_1, and x_2), we determined the fault detection capability of each test case using the IRCOV model (see Eq. 4). Finally, we arranged the test cases in descending order by their fault detection ability.

Evaluating the technique: After having a prioritized set of test cases, we ran them on the 50 faulty versions of each fold we created using the test data set of mutants. To evaluate the results, we used the average percentage of fault detection (APFD) (see Eq. 7).

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_N}{nm} + \frac{1}{2n} \tag{7}$$

4.4 Analysis of the replication results

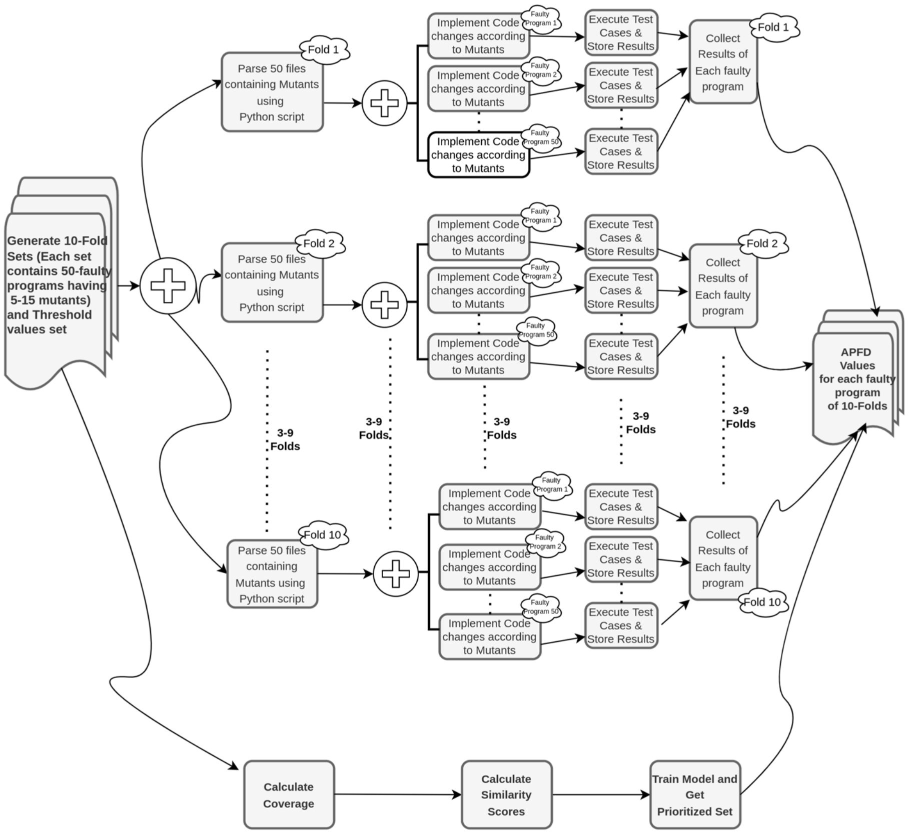
We implemented IRCOV for line, branch, and method coverage. As described above, we calculated the APFD values for all coverage types for each fold and captured the intermediate results (see Table 2).

We translated the APFD values for Commons-CLI from the original study to compare the replication and the original study results. Then we plotted the APFD values of the original and replication study in the box plot, a statistical tool to visually summarize and compare the results of two or more groups (Do et al., 2010; Williamson et al., 1989). Boxplots of APFD values enabled us to compare the replication and original study results visually.

To triangulate our conclusions, we applied hypothesis testing. We used Wilcoxon signed-rank test to compare the IRCOV original and IRCOV replication results. Also, the original study Kwon et al. (2014) used Wilcoxon signed-rank test to compare the IRCOV results with the baseline methods. Wilcoxon signed-rank test is suitable for paired samples where data is the outcome of before and after treatment. It measures the difference between the median values of paired samples (Gibbons, 1993). We were interested in measuring the difference between the median APFD values of IRCOV original and IRCOV replication. Therefore, the appropriate choice to test our results was Wilcoxon signed-rank test.

We tested the following hypothesis:

H_{0LC}: There is no significant difference in the median APFD values of the original and replication studies using line coverage.



Generate 10-Fold Sets (Each set contains 50-faulty programs having 5-15 mutants) and Threshold values set
Parse 50 files containing Mutants using Python script
Fold 1
Fold 2
Fold 10
Implement Code changes according to Mutants
Faulty Program 1
Faulty Program 2
Faulty Program 50
Execute Test Cases & Store Results
Collect Results of Each faulty program
3-9 Folds
APFD Values for each faulty program of 10-Folds
Calculate Coverage
Calculate Similarity Scores
Train Model and Get Prioritized Set

Fig. 2 Steps to automate the replication of IRCOV

H_{0BC}: There is no significant difference in the median APFD values of the original and replication studies using branch coverage.

H_{0MC}: There is no significant difference in the median APFD values of the original and replication studies using method coverage.

4.5 Automation of replication

The replication was implemented using Python scripts. They are available (Minhas & Irshad, 2021). Figure 2 presents the details of automation steps for the replication of IRCOV. The original study's authors proposed that ten-fold-based execution is needed (when historical data is not available) to evaluate their original technique. Therefore, our implementation (fold_generator) (Minhas & Irshad, 2021) generates ten folds of the object program at the first stage. Thereafter, it generates 50 faulty versions of each fold, whereas each version contains 5–15 mutants (faults). After generating the faulty versions, the script makes the corresponding changes in the code. Finally, the tests are executed, and their results are extracted. Later, using the test results, we calculated each fold's APFD values.

5 Results

This section presents the findings from the replication. The results are organized according to research questions listed in Sect. 4.

5.1 RQ1. The degree to which the replication is feasible to implement

The first goal was to replicate the IRCOV technique with the four software programs described in the original study (Kwon et al., 2014) (RQ1.1) and with two additional software programs (RQ1.2).

Concerning RQ1.1, out of four software programs used in the original study, we could completely replicate the IRCOV technique with only Commons-CLI. Whereas, with the other three software programs, (i) XML Security, (ii) Commons-Collections (CCN), and (iii) Joda-Time (JOD), the replication was partially successful or unsuccessful. Concerning RQ1.2, we could not successfully replicate the IRCOV with these additional software programs (i) Commons-Email and (ii) Log 4j. The details of completely successful, partially successful, and unsuccessful cases are discussed in the subsequent paragraphs.

Successful replication implementation: We successfully replicated IRCOV with Commons-CLI. After going through the steps presented in Sect. 4.3.2, for every fold, we were able to calculate the respective coverage information and similarity score of each test case. Table 2 presents the intermediate results for the replication of IRCOV with Commons-CLI. These include training error, chosen value of IDF threshold, regression coefficient θ_0, coverage weight θ_1, and weight for similarity score θ_2.

To evaluate the performance of IRCOV, we have calculated APFD values for all ten folds of each coverage type (branch, line, and method) (see Table 3). The APFD value ranges from 0.547 to 0.873 for branch coverage, whereas the average (median) APFD value for branch coverage is 0.747. The APFD values for line coverage range from 0.609 to 0.873, and the average APFD value for line coverage is 0.809. Finally, the APFD value for method coverage ranges from 0.549 to 0.864, and the average APFD for method coverage is 0.772. These results show that the IRCOV model performed best for line coverage as the mean APFD for line coverage is highest among the coverages.

Table 3 APFD values for all ten folds of each coverage type

Folds	Branch coverage	Line coverage	Method coverage
Fold 1	0.874	0.874	0.865
Fold 2	0.816	0.866	0.790
Fold 3	0.646	0.643	0.613
Fold 4	0.757	0.816	0.755
Fold 5	0.725	0.715	0.721
Fold 6	0.796	0.829	0.829
Fold 7	0.841	0.839	0.839
Fold 8	0.610	0.610	0.585
Fold 9	0.548	0.622	0.594
Fold 10	0.736	0.803	0.803

Partial or unsuccessful replication: Our first unsuccessful replication was concerning XML Security. We did not find all the program versions used in the original study (Kwon et al., 2014). Therefore, we decided to use the versions that have slightly similar major/minor release versions. We downloaded available XML Security versions 1, 1.5, and 2.2.3. The first two downloaded versions (version 1 and version 1.5) were not compiling due to the unavailability of various dependencies. The logs from the compilation failures are placed in the folder "LogsXmlSecurit" available at Minhas and Irshad (2021).

We were able to compile the third XML Security version 2.2.3, but we could not continue with it because this version contained several failing test cases (see Minhas & Irshad, 2021). With already failing test cases, training the model correctly and getting the appropriate list of prioritized test cases was difficult.

The second and third attempts were made on Commons-Collection and Joda-Time. Compared to other projects used in the replication experiment, these projects contain more test classes (see Table 1). The mutants were generated for each of these projects. Out of 50 faulty versions, the first faulty version of Commons-Collection (Fold1) took 40 min in execution, and similarly, the first faulty version of Joda-Time (Fold1) took 36 min in execution. This was a limitation since, to train the model, we had to use ten folds, and for each fold, we had to execute 50 faulty versions containing five to fifteen faults in each faulty version. Our estimate shows that it would take 2000 min to train Commons-Collection and 1810 min to train Joda-Time. It would take 64 h (approx) before we get the final results of these two programs – provided that the Internet connection remains stable and working. We made several attempts to perform replications on these two projects (Joda-Time and Common-Collection). However, technical limitations limit the number of times these projects could run or compile. These limitations were:

- Maven-based projects fetch the dependencies from the maven-repositories. These projects were fetching huge amounts of data for each execution cycle. Our client was banned for short duration by Maven repositories because of sending too many requests in a short amount of time.
- Since these projects download libraries from the internet during each compile-run cycle, the internet connections (at three different places) did not keep up with the amount of data continuously requested by the client. A simple disconnection means that the whole script will stop working.

Furthermore, parsing and analyzing the data of many test classes is also time-consuming. Due to these technical and resource limitations, we abandoned the replications for these two projects. Furthermore, parsing and analyzing the data of many test classes is also time-consuming. Due to these technical and resource limitations, we abandoned the replications for these two projects. We have further discussed these issues in Sect. 6.

The fourth unsuccessful replication attempt was executed on Commons-Email. This time the replication was unsuccessful because of faulty mutants generated by the mutant software. For instance, it suggested replacing variable names with 'null' (see Listing 1 & 2). The actual code was *this.name = null*; while after mutant injection, the code turned to *this.null = null*.

Another type of faulty instance was when MAJOR suggested modifying a line in the code that resulted in Java compilation errors (such as "unreachable statement"). There were several such faulty mutants that made the program fail to compile, and hence no further processing was possible. The detail of all faulty mutants is available in the folder "CommonsEmail" at Minhas and Irshad (2021).

```
35:EVR:<IDENTIFIER(java.lang.String)>:<DEFAULT>:org.apache.commons.mail.
ByteArrayDataSource@setName(java.lang.String):214:name |==> null
```

Listing 1 Faulty mutant generated by the tool

Listing 2 Code generated after the insertion of faulty mutant

```
public void setName(final String name)
  {
  // this.name = null;
       this.null = null; }
```

We also made unsuccessful attempts to change the mutant generator to rectify this problem. However, each mutant generator presented a new set of problems. The lessons learned from the usage of different mutant generators are described in the next section.

The fifth replication attempt was executed on Log4j. We followed all the steps (using our automated scripts) proposed by the authors of the original study. We successfully generated the mutants for this program. However, the replication was stopped at the point when the steps to train the model failed. The proposed approach in the original study is based on the coverage information of each code class and test class. This time the low coverage of the test cases caused the issue. During the training of the model, we realized that because of the low coverage of the test cases, we were unable to calculate the values of regression coefficients, and as a result, we could not generate the prioritized set of test cases. We developed a Jupyter notebook to describe each step of this partially successful replication (see Minhas & Irshad, 2021). Compared to the other software programs selected in this study, with 169646 LOC, Log4j is a large program. Thus, a lot of time was needed to train the model for Log4j. For all ten folds, with 50 faulty versions of each fold and with five to fifteen faults in each faulty version, it required approximately 60 h to train the model.

Key findings: Concerning RQ1, the replication was only feasible in one of six cases; the key reasons are listed below.

1. The inability to use the system under test was caused by compatibility issues (unavailability of system versions and dependencies).
2. Already failing test cases made the replication fail.
3. Mutant generators created issues in running the replication, and workarounds were difficult to implement.
4. Test cases require a certain level of coverage to train the model.
5. More effort is required to train the model for large-sized software programs.

5.2 RQ2. Comparison of the results to the original study

We could replicate IRCOV with only one program (Commons-CLI) out of six selected programs. Therefore results presented for RQ2 may be considered partial as these are based on the outcome of a single execution of IRCOV.

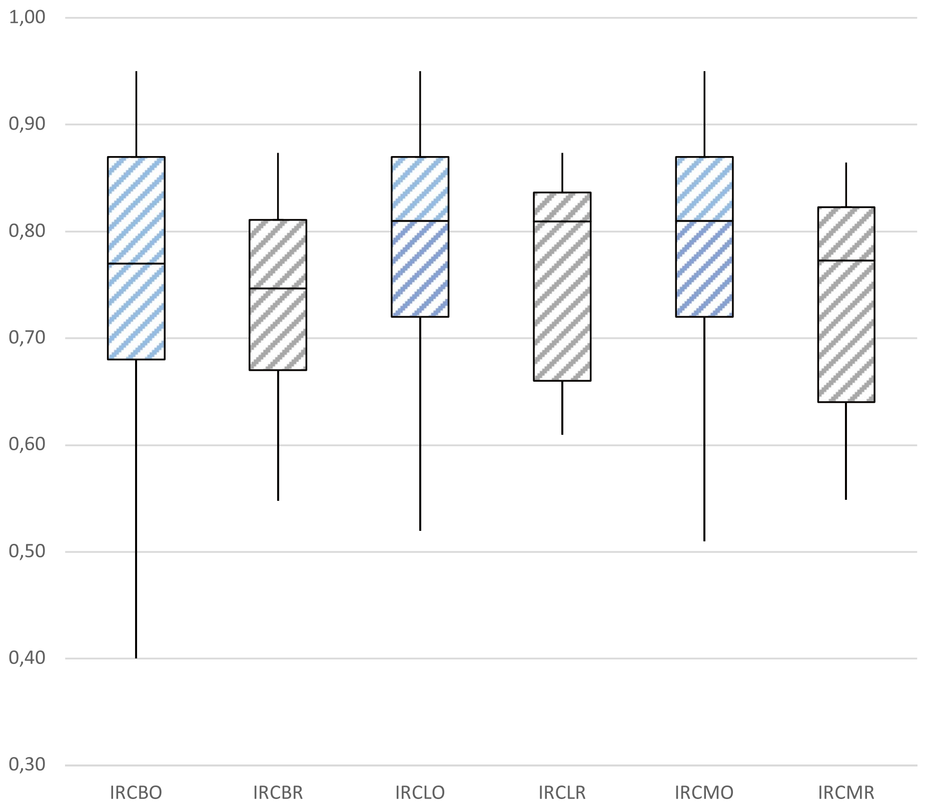


Fig. 3 APFD Boxplots for IRCOV Original vs IRCOV Replication (IRCBO= IRCOV Branch coverage original, IRCBR= IRCOV Branch Coverage Replication. IRCLO= IRCOV Line coverage original, IRCLR= IRCOV Line coverage replication. IRCMO= IRCOV Method coverage original, IRCMR=IRCOV Method coverage replication

For comparison purposes, we relied on what has been reported by the authors of the original study, as that was the only source to reach and reconstruct the results. Figure 3 presents the APFD boxplots of the original and replication study for Commons-CLI. Boxplots with blue patterns represent the original study results, and boxplots with gray patterns represent the replication study results. We can see that in all cases, the APFD values of the original study are slightly better compared to the values of the replication. We applied statistical tests to detect whether the results of the replication and the original study differed.

We applied Wilcoxon singed-rank test to compare the replication results for branch, line, and method coverage of Commons-CLI with the original study's results. The results are significant if the *p*-value is less than the level of significance (Du Prel et al., 2009). The difference between the two implementations would be significant if the *p*-value is less than 0.05.

Table 4 presents the results of the statistical test. The *p*-value for branch coverage is 0.475, which is greater than 0.05 (significance level). Therefore, we can not reject the null hypothesis. That means we can not show a significant difference in the APFD values for branch coverage of Commons-CLI between the replication and the original study.

Similarly, the *p*-value for line coverage is 0.415, greater than the set significance level. Based on the statistical results, we can not reject the null hypothesis. This implies that we

Table 4 Statistical results of replication compared to the original study for Commons-CLI

Coverage	α	p-value	95% Conf. Int.
Branch	0.05	0.475	0.646–0.816
Line	0.05	0.415	0.668–0.845
Method	0.05	0.103	0.652–0.827

can not show a significant difference in the APFD values for the line coverage of Commons-CLI between the replication and the original study.

Finally, the p-value for method coverage is 0.103. Based on this result, we can not reject the null hypothesis. Therefore no significant difference in the APFD values for the method coverage of Commons-CLI between the replication and the original study.

From the t-test results, we can conclude that for all three coverage types (branch, line, and method), we did not find any significant difference between the replication and the original study. Therefore, we can state that the replication experiment did not deviate from the original result to the degree that would lead to the test detecting a significant difference.

Key findings: Concerning RQ2, we compared the replication results of the successful case (i.e., Commons-CLI) with the original study's results. Below are the key findings for RQ2.

1. The statistical test did not detect a significant difference in the APFD values of the replication and the original study concerning the three coverage measures investigated.
2. We conclude that the results of the original study are verifiable for Commons-CLI.

6 Discussion

6.1 Lessons learned of replicating artifact-based studies in software testing

We replicated the study presented in Kwon et al. (2014) to promote artifact-based replication studies in software engineering, validating the correctness of the original study and exploring the possibilities of adopting regression testing research in the industry.

Overall, it is essential to capture and document assumptions and constraints concerning the techniques to be replicated, as well as the conditions for running replication. We highlight several factors of relevance that were observed.

Conditions concerning System under Test (SuT) complexity: From the replication results, we learned that the technique (IRCOV) presented in (Kwon et al., 2014) is replicable for small and medium software programs provided the availability of context information, as it was demonstrated by replication Commons-CLI. However, we could not demonstrate the replication of IRCOV with other small and medium programs (e.g., XML Security) because of the availability of context information. Similarly, we faced issues while replicating IRCOV with comparatively large software programs having more test classes (e.g., Commons-Collection and Joda-Time). Furthermore, we did not replicate IRCOV on any other industry-strength software. The industry-strength software may be written in different languages, frameworks, and coding conventions (e.g., size of class, number of classes). The technique may produce different results if different types of programming languages are used. Furthermore, industry-strength software is continuously changing due to maintenance needs. This will result in running the approach each time a change is introduced. In short, further evaluation is required before claiming the generalizability of IRCOV concerning systems under test.

Based on our experiences during the replication of IRCOV, we can say that with its current guidelines, the technique is challenging to implement with large software programs because it requires significant effort to train the model. The restriction of 10-folds, 50 faulty versions for every fold, and 5 to 15 faults in every faulty version would require a substantial effort. For example, while attempting to replicate the original study with Commons-Collection and Joda-Time, we estimated it would take approximately 64 h to train the model for each software program. This limitation can be managed by reducing the number of faulty versions for each fold, but this may degrade the accuracy and increase the training error.

Conditions concerning the characteristics of the test suite: Test cases available with Log4j have low coverage, limiting the chance of correctly training the model and generating a reasonable prioritization order of the test cases. Coverage is one of the primary inputs required for the test case prioritization using the IRCOV model. Another problem we encountered was the presence of already failing test cases in one of the versions of XML Security. Test cases are used to calculate the coverage score and similarity scores of the project. If a handful of test cases fail (as in XML Security version 2.2.3), wrong coverage information and similarity scores are calculated. This results in the wrong prioritization of test cases as well as faulty training of the model (which is used to identify prioritized test cases). Another drawback with failing test cases concerns the use of mutations. If tests are already failing and when mutants are introduced, then the effectiveness is unreliable as tests are already failing because of other issues. Further conditions may be of relevance in studies focusing on different aspects of software testing. Here, we will highlight how important it is to look for these conditions and document them. This is also relevant for practice, as it demonstrates under which conditions a technique may or may not be successful.

Availability of experimental data for artifact-based test replications: One of the constraints regarding the replicability of the IRCOV technique is the availability of experimental data. For example, the authors of the original study (Kwon et al., 2014) stated that they used in-house built tools to conduct the experiment, but they did not provide any source of these tools and also did not include details of the automation tools. Therefore, setting up the environment to replicate IRCOV with the first program took a significant effort. There are various limitations concerning the data sets and tools required to work with the recommended steps. Regarding data sets, we have recorded the findings in Sect. 5. These include the compatibility of SIR artifacts. For example, we faced difficulties working with XML Security version 1 because of various dependencies. While working with version 2.2.3 of XML Security, we encountered errors in the version. Therefore, we could not collect the coverage information. Ultimately, we were unable to replicate the technique with any of the versions of XML Security.

Reflections on mutant generators: In the absence of failure data, the authors of the original study suggested using mutation faults, and they used the MAJOR mutation tool to generate the mutants. In one of our cases (Commons-Email), the mutation tool generated inappropriate mutants that led to the build failure. Therefore, no further progress was possible with this case. One option could be removing the faulty mutants and including only the correct ones. In our opinion, removing faulty mutants may bring different results from the original study, thus limiting the validity of replication.

To overcome the difficulty with the replication Commons-Email, we tried different open-source mutation generators available. Each presented various benefits and challenges documented in Table 5. After trying out different mutation tools, we learned that among the available options, MAJOR is an appropriate tool for Java programs, as it generates the mutants dynamically.

Reflections on the requirements of the experimental setup: We faced limitations while attempting the replication of the technique with larger software programs. The time required to train the model with larger programs was longer. We do not know whether the original authors experienced similar problems since the original study did not discuss similar issues. We suggest that original studies explicitly report any special experimental setup requirements (e.g., hardware, software, etc.). This will help replicators (researchers and practitioners) to replicate/adapt the studies to a new context. Further, collaboration between the original authors and replicators may help resolve such issues. However, as mentioned in the earlier sections, we could not establish an effective collaboration with the authors of the original study.

Reflections on the IRCOV technique: Besides the limitations highlighted earlier, based on the replication results of Commons-CLI, we can state that the IRCOV technique can be replicated for smaller software programs provided the availability of required information. Though we could not replicate the technique on all the selected programs, the replication results of Commons-CLI showed that the original authors' claim about the performance of the IRCOV was verifiable. The IRCOV can be a valuable technique from an industry perspective because it focuses on prioritizing test cases and detecting faults in less-tested code while taking coverage of test cases into account during the prioritization process. Besides increasing the test suite's rate of fault detection, it can help the practitioners work with one of their goals (e.g., controlled fault slippage). However, the technique needs to be modified to decrease the time required for programs with many test classes. Looking at regression testing in practice, the practitioners recognize and measure the coverage metric (Minhas et al., 2020). Failure history is the only information that companies need to maintain. In the presence of actual failure data, we do not need to use the mutants to train the IRCOV model extensively, and we can reduce the number of faulty versions for each fold and the number of folds.

Pursuing the first research question (RQ1) provided us with a deeper insight into the various aspects and challenges of external replication. The lessons learned in this pursuit are interesting and provide recommendations in the context of replication studies in software engineering. The existing literature revealed that the trend of replication studies in software engineering is not encouraging (Da Silva et al., 2014). The studies report that internal replications are much higher than external replications (Bezerra et al., 2015; Da Silva et al., 2014). While searching the related work, we observed that external replications in the software testing domain are few compared to internal replications. There could be several reasons for the overall lower number of replication studies in software engineering, but we can reflect on our experiences concerning external replications as we have undergone an external replication experiment.

One issue we would like to highlight is the substantial effort needed to implement the replication. Replication effort can be substantially reduced with more detailed documentation of the original studies, the availability of appropriate system versions

Table 5 Comparison of mutant generators

No	Mutation tool	Benefits	Challenges
1	Major[a]	(i) Easy to use. (ii) Most commonly used mutant generator	(i) Faulty mutant generated. (ii) Needs upgrade to latest Java versions. (iii) Documentation needs improvement
2	µJava[b]	(i) IDE plugin available. (ii) User decides what types of mutants can be generated	(i) Exporting mutants separately is not supported. (ii) Does not support latest Java versions (iii) GUI often crashes while generating mutants
3	Jester[c]	Two types of Jester versions, a complete version and a simple version	Latest update was more than 10 years ago. We were unable to generate mutations or start the program despite following all steps
4	Jumble[d]	(i) Support recent Java versions. (ii) Integration with IDE Supported	Unable to generate mutants despite the following examples. The latest update was 6 years ago
5	PIT[d]	The most recent and complete mutant generator. Mutants are generated, and tests are executed. A report is generated for the user	(i) Unable to export the mutants. (ii) Lack of diversity in the mutants. (iii) Each execution produced exact same mutants

[a]https://mutation-testing.org/

[b]https://cs.gmu.edu/offutt/mujava/

[c]http://jester.sourceforge.net/

[d]http://jumble.sourceforge.net/

[e]https://pitest.org/

and their dependencies, and the knowledge about prerequisites and assumptions. Better documentation and awareness of conditions may facilitate a higher number of replications in the future.

6.2 General lessons learned for artifact-based replications

Table 6 provides an overview of the challenges we encountered during the replications. It lists the possible impact of each challenge on the results of replication, and the table also presents a list of recommendations for researchers. The following provides a brief discussion of the lessons learned in this study.

Documenting the original experiment: The authors of the original studies need to maintain and provide comprehensive documentation concerning the actual experiment. The availability of such documents will help the independent replicators understand the original study's context. In the absence of such documentation, the replicators need to invest more effort to understand the original study's context. In this regard, we suggest using open-source repositories to store and publish the documentation. The documentation may contain the detail of the experimental setup, including the tools used to aid the original experiment, automation scripts (if used/developed), and the internal and final results of the study. Furthermore, the authors can also include detail about any special requirements or considerations that need to be fulfilled for the successful execution of the experiment.

Collaboration with the original authors: Because of page limits posed by the journals and conferences, every aspect of the study can not be reported in the research paper. Sometimes, the replicators need assistance from the original authors regarding any missing aspect of the study. Therefore, it is essential that in case of any such query from the replicators, the original study's authors should be able to assist them. Such cooperation can promote replication studies in software engineering. In our opinion, lack of collaboration is one reason for fewer replication studies in software engineering. However, it is important to still conduct the replications as independently as possible due to possible biases (i.e., avoiding turning an external replication into an internal one).

Maintaining open-source repositories: Open-source repositories (one example being SIR) provide an excellent opportunity for researchers to use the data sets while conducting software engineering experiments. A large number of researchers have benefited from these data sets. We learned that some of the data sets available in repositories are outdated and need to be maintained. Such data sets are not helpful, and studies conducted using these data sets would be complex to adopt/replicate. It is, therefore, essential that authors explicitly state the versions they used in their own studies. In addition, we recommend that authors of original studies as well as replications ensure that the dependencies or external libraries are stored to facilitate the replications of the system under test.

Tools compatibility: In many cases, the authors need to use open-source tools to assist in the execution of their experiment. Such tools need to be well-maintained and updated. In case of compatibility issues, these tools can hinder the replication process. For example, the study we replicated uses a mutation tool (MAJOR). Although it is one of the best available options, the tool generated inappropriate mutants for one of our cases due to some compatibility issues. Ultimately, after a significant effort, we had to abandon the replication process for that case. Here, we also would like to highlight that one should document the versions of the tools and libraries used (also including scripts written by the researchers - e.g., in Python).

Table 6 Recommendations drawn from the challenges/lessons learned

Challenge	Impact	Recommendation
Documentation of the original experimental setup	Replicators have to invest additional effort to understand the context of the study	Original authors need to maintain/publish a comprehensive documentation of experimental setup
Collaboration with the authors of original studies	Absence of experimental data and support from original authors can complicate the replication process	In the event of a request from the replicators, the authors of the original study provide assistance in the form of essential information regarding the original experiment
Issues with the open-source data set	Replication experiments may fail due to these issues	Open-source repositories need to be maintained and up to date
The system under Test (SuT) and tools compatibility issues	Any compatibility issue of the tools required to replicate the original experiment can create a bottleneck for the replication	Such tools (e.g., Mutation tools in our case) need to be maintained to make them compatible with new development frameworks. The same applies to the system under test

Documenting successes and failures in replications: Besides the significance of documenting every aspect of the original experiment, recording every single event of replication (success & failure) is critical for promoting future replications and industry adoptions of research. We recommend storing the replication setups and data in open-source repositories and providing the relevant links in the published versions of the articles.

Automation of replication: A key lesson learned during the replication of the original study is that the documentation of the setup and execution of replication could be automated with the help of modern tools and programming languages. This automation will help in reproducing the original results and analysis for researchers reviewing or producing the results from the studies. We have provided programming scripts that describe and document all the steps (and the consequences of these steps).

7 Threats to validity

7.1 Internal validity

Internal validity refers to the analysis of causal relations of independent and dependent variables. In our case, we have to see if the different conditions affect the performance of IRCOV. IRCOV depends upon two inputs, coverage of each test case and a similarity score calculated based on TF-IDF. We used the test cases available within the software programs. Therefore, we do not have any control over the coverage of these test cases. However, the choices of mutants can impact the similarity score. To avoid any bias, we generated the mutants using a tool and used a random generator to select the mutants for different faulty versions of the software programs. Furthermore, we trained IRCOV sufficiently before applying it to test data by following the ten-fold validation rule. Since we measured the performance of IRCOV using the APFD measure, the results of the successful case were not significantly different from the original study's results. Therefore, we can argue that our treatment did not affect the outcome of IRCOV. Hence minimizing the threats to internal validity.

7.2 Construct validity

Construct validity is concerned with the underlying operational measures of the study. Since it is a replication study and we followed the philosophy of exact replication (Shull et al., 2008) if the original study suffers from any aspects of construct validity, the replication may do so. For instance, the use of mutation faults could be a potential threat to the construct validity because of the following two reasons:

- Mutation faults may not be representative of real faults.
- Possible researchers' bias concerning the nature of mutation faults.

Concerning the first reason, the use of mutation faults to replace the real faults is an established practice, and researchers claim that mutation faults produce reliable results and hence can replace the real faults (Andrews et al., 2005; Do & Rothermel, 2006). We used an automated mutation tool to generate the mutants to avoid bias. Also, to select the mutants for validation, training, and test set, we used an automated random selector. Hence no human intervention was made during the whole process. Furthermore, we discussed the strengths and weaknesses of different tools.

7.3 External validity

External validity is the ability to "generalize the results of an experiment to industrial practice" (Wohlin et al., 2012). The software programs used in the replication study are small and medium-sized Java programs. Therefore, we can not claim the generalizability of results to large-scale industrial projects. The results produced in replication for one program (Commons-CLI) conform with the original study's results. However, we could not demonstrate the use of the technique for the other programs used in the original study and on the additional software programs. Therefore, there is a possibility of threats to the external validity of the replication study.

8 Conclusions and future work

This study reports the results of a replication experiment to evaluate a test case prioritization technique using information retrieval (IR) concepts proposed initially by Kwon et al. We attempted to replicate the original study using six Java programs: Commons-CLI, XML Security, Commons-Collection, Joda-Time, Commons-Email, and Log4j. In the first research question (RQ1), the aim was to see if the technique is replicable, and in the second research question (RQ2), we aimed to see if the replication results conform to the ones presented in the original study.

We have faced various challenges while pursuing RQ1. These challenges are related to the availability of the original experimental setup, collaboration with the original authors, system under test, test suites, and compatibility of support tools. We were able to replicate the technique only with Commons-CLI, which is a smaller program. Based on our experience, we can conclude that the technique is replicable for small software programs (such as Commons-CLI) subject to the availability of required information. However, it is hard to implement the technique with larger software programs because it requires a substantial effort to implement it for a larger program. Concerning RQ1, the important concluding point is that it is not feasible to externally replicate an experiment when context information and relevant data are not available. Furthermore, without the support of the original authors, it becomes a challenging task.

To verify the original study's results (RQ2), since we could replicate IRCOV with only one program, Commons-ClI, we compared the replication results for Commons-CLI with those presented in the original study. These results validated the original study's findings as the statistical test confirms no significant difference between the APFD values of the replication and the actual experiment. However, we may say that there are limitations in our results being partially conformed with the original study because we could not replicate the technique with all selected artifacts due to missing dependencies, broken test suites, and other reasons highlighted earlier.

The technique can be helpful in the industrial context as it prioritizes the test cases that target the less tested code. It can help the practitioners to control fault slippage. However, it needs some improvements in training and validation aspects to scale the technique to the industry context. To support the future replications/adoption of IRCOV, we have automated the IRCOV steps using Python (Jupyter notebook). Lessons learned during the IRCOV replication are essential for the original authors for reporting and documenting the experimental setup. Similarly, for replicators (researchers and practitioners), these lessons are essential to learning about the requirements to replicate a technique in a new environment.

We plan to work with more artifacts with actual faults to test the technique's (IRCOV) effectiveness in the future, and we plan to see the possibilities of scaling it up for larger projects. In addition to that, we want to evaluate our proposed guidelines (under lessons learned) using different studies from industrial contexts.

Author contributions Nasir Mehmood Minhas and Kia Petersen conceived the idea of the study, while Nasir Mehmood Minhas and Mohsin Irshad carried out the replication experiment. Kia Petersen and Jürgen Börstler reviewed the implementation steps and challenged the implementation wherever necessary. In Short, all authors have contributed to different phases of this study, i.e., the conception of the idea, implementation, and manuscript writing. All authors read and approved the final manuscript, and they stand accountable for all aspects of this work's originality and integrity.

Funding Open access funding provided by Blekinge Institute of Technology. This work has in parts been supported by ELLIIT: the Swedish Strategic Research Area in IT and Mobile Communications.

Data availability The datasets and automated scripts generated during the current study are available in the Mendeley Data repository and can be accessed using the link given in (Minhas & Irshad, 2021).

Declarations

Competing interest The authors declare no competing interests.

References

Ali, N. B., Engström, E., Taromirad, M., Mousavi, M. R., Minhas, N. M., Helgesson, D., Kunze, S., & Varshosaz, M. (2019). On the search for industry-relevant regression testing research. *Empirical Software Engineering*, 1–36.

Amati, G. (2009). *Information retrieval models*. Springer, New York, NY. pp. 1523–1528. https://doi.org/10.1007/978-1-4614-8265-9_916

Andrews, J. H., Briand, L. C., & Labiche, Y. (2005). Is mutation an appropriate tool for testing experiments? In: Proceedings of the 27th International Conference on Software Engineering, pp. 402–411.

Bajaj, A., & Sangwan, O. P. (2019). A systematic literature review of test case prioritization using genetic algorithms. *IEEE Access, 7*, 126355–126375.

Bezerra, R. M., da Silva, F. Q., Santana, A. M., Magalhaes, C. V., & Santos, R. E. (2015). Replication of empirical studies in software engineering: An update of a systematic mapping study. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE. pp. 1–4.

Carver, J. C. (2010).Towards reporting guidelines for experimental replications: A proposal. In: 1st International Workshop on Replication in Empirical Software Engineering, Citeseer, vol 1. pp. 1–4.

Catal, C. (2012). On the application of genetic algorithms for test case prioritization: A systematic literature review. In: Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies, ACM. pp. 9–14.

Catal, C., & Mishra, D. (2013). Test case prioritization: A systematic mapping study. *Software Quality Journal, 21*(3), 445–478.

Chen, Z., Guo, H. F., & Song, M. (2018). Improving regression test efficiency with an awareness of refactoring changes. *Information and Software Technology, 103*, 174–187.

Chi, J., Qu, Y., Zheng, Q., Yang, Z., Jin, W., Cui, D., & Liu, T. (2020). Relation-based test case prioritization for regression testing. *Journal of Systems and Software, 163*, 110539.

Da Silva, F. Q., Suassuna, M., França, A. C. C., Grubb, A. M., Gouveia, T. B., Monteiro, C. V., & dos Santos, I. E. (2014). Replication of empirical studies in software engineering research: A systematic mapping study. *Empirical Software Engineering, 19*(3), 501–557.

Dahiya, O., & Solanki, K. (2018). A systematic literature study of regression test case prioritization approaches. *International Journal of Engineering & Technology, 7*(4), 2184–2191.

de Magalhães, C. V., da Silva, F. Q., Santos, R. E., & Suassuna, M. (2015). Investigations about replication of empirical studies in software engineering: A systematic mapping study. *Information and Software Technology, 64*, 76–101.

Do, H., Rothermel, G., & Kinneer, A. (2004). Empirical studies of test case prioritization in a Junit testing environment. In: 15th International Symposium on Software Reliability Engineering, IEEE. pp. 113–124.

Do, H., Elbaum, S., & Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering, 10*(4), 405–435.

Do, H., Mirarab, S., Tahvildari, L., & Rothermel, G. (2010). The effects of time constraints on test case prioritization: A series of controlled experiments. *IEEE Transactions on Software Engineering, 36*(5), 593–617.

Do, H., & Rothermel, G. (2006). On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering, 32*(9), 733–752.

Du Prel, J. B., Hommel, G., Röhrig, B., & Blettner, M. (2009). Confidence interval or p-value?: Part 4 of a series on evaluation of scientific publications. *Deutsches Ärzteblatt International, 106*(19), 335.

Ekelund, E. D., & Engström, E. (2015). Efficient regression testing based on test history: An industrial evaluation. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution, ICSME. pp. 449–457.

Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering, 28*(2), 159–182.

Engström, E., & Runeson, P. (2010). A qualitative survey of regression testing practices. In: Proceedings of the 11th International Conference on Product-Focused Software Process Improvement PROFES. pp. 3–16.

Engström, E., Runeson, P., & Skoglund, M. (2010). A systematic review on regression test selection techniques. *Information & Software Technology, 52*(1), 14–30.

Fang, H., Tao, T., & Zhai, C. (2004). A formal study of information retrieval heuristics. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 49–56.

Felderer, M., & Fourneret, E. (2015). A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer, 17*(3), 305–319.

Gibbons, J. D. (1993). Location tests for single and paired samples (sign test and Wilcoxon signed rank test).

Harrold, M. J., & Orso, A. (2008). Retesting software during development and maintenance. In: Proceedings of the Frontiers of Software Maintenance Conference. pp. 99–108.

Hasnain, M., Ghani, I., Pasha, M. F., Malik, I. H., & Malik, S. (2019). Investigating the regression analysis results for classification in test case prioritization: A replicated study. *International Journal of Internet, Broadcasting and Communication, 11*(2), 1–10.

ISO/IEC/IEEE. (2017). International standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*. pp. 1–541. https://doi.org/10.1109/IEEESTD.2017.8016712

Ivarsson, M., & Gorschek, T. (2011). A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering, 16*(3), 365–395.

Juristo, N., & Gómez, O. S. (2012). Replication of software engineering experiments. Springer Berlin Heidelberg. pp. 60–88. https://doi.org/10.1007/978-3-642-25231-0_2

Just, R. (2014). The major mutation framework: Efficient and scalable mutation analysis for java. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. pp. 433–436.

Just, R., Schweiggert, F., & Kapfhammer, G. M. (2011). Major: An efficient and extensible tool for mutation analysis in a java compiler. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), IEEE. pp. 612–615.

Kazmi, R., Jawawi, D. N. A., Mohamad, R., & Ghani, I. (2017). Effective regression test case selection: A systematic literature review. *ACM Computing Surveys,50*(2), 29:1–29:32.

Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., & Tumeng, R. (2018). Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology, 93*, 74–93.

Krein, J. L., & Knutson, C. D. (2010). A case for replication: Synthesizing research methodologies in software engineering. In: RESER2010: Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research, Citeseer. pp. 1–10.

Kwon, J. H., Ko, I. Y., Rothermel, G., & Staats, M. (2014). Test case prioritization based on information retrieval concepts. In: 2014 21st Asia-Pacific Software Engineering Conference, IEEE, vol 1. pp. 19–26.

Legunsen, O., Hariri, F., Shi, A., Lu, Y., Zhang, L., & Marinov, D. (2016). An extensive study of static regression test selection in modern software evolution. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 583–594.

Legunsen, O., Shi, A., & Marinov, D. (2017). Starts: Static regression test selection. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 949–954.

Lima, J. A. P., & Vergilio, S. R. (2020). Test case prioritization in continuous integration environments: A systematic mapping study. *Information and Software Technology, 121*, 106268.

Minhas, N. M., & Irshad, M. (2021). Data set used in the replication of an IR based test case prioritization techniques (IRCOV). https://data.mendeley.com/drafts/ccnzpxng54, https://doi.org/10.17632/ccnzpxng54.1

Minhas, N. M., Petersen, K., Ali, N., & Wnuk, K. (2017). Regression testing goals-view of practitioners and researchers. In: 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), IEEE. pp. 25–32.

Minhas, N. M., Petersen, K., Börstler, J., & Wnuk, K. (2020). Regression testing for large-scale embedded software development - Exploring the state of practice. *Information and Software Technology, 120*, 106254. https://doi.org/10.1016/j.infsof.2019.106254

Ouriques, J. F. S., Cartaxo, E. G., & Machado, P. D. (2018). Test case prioritization techniques for model-based testing: A replicated study. *Software Quality Journal, 26*(4), 1451–1482.

Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: A systematic literature review. *Empirical Software Engineering, 27*(2), 1–43.

Pannu, M., James, A., & Bird, R. (2014). A comparison of information retrieval models. In: Proceedings of the Western Canadian Conference on Computing Education. pp. 1–6.

Peng, Q., Shi, A., & Zhang, L. (2020). Empirically revisiting and enhancing IR-based test-case prioritization. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 324–336.

Pittelkow, M. M., Hoekstra, R., Karsten, J., & van Ravenzwaaij, D. (2021). Replication target selection in clinical psychology: A bayesian and qualitative reevaluation. *Clinical Psychology: Science and Practice, 28*(2), 210.

Qiu, D., Li, B., Ji, S., & Leung, H. K. N. (2014). Regression testing of web service: A systematic mapping study. *ACM Computing Surveys,47*(2), 21:1-21:46.

Rainer, A., & Beecham, S. (2008). A follow-up empirical evaluation of evidence based software engineering by undergraduate students. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. pp. 78–87.

Rainer, A., Jagielska, D., & Hall, T. (2005). Software engineering practice versus evidence-based software engineering research. In: Proceedings of the ACM Workshop on Realising Evidence-based Software Engineering (REBSE '05). pp. 1–5. https://doi.org/10.1145/1082983.1083177

Roelleke, T. (2013). Information retrieval models: Foundations and relationships. *Synthesis Lectures on Information Concepts, Retrieval, and Services, 5*(3), 1–163.

Rosero, R. H., Gómez, O. S., & Rafael, G. D. R. (2016). 15 years of software regression testing techniques - A survey. *International Journal of Software Engineering and Knowledge Engineering, 26*(5), 675–690.

Saha, R. K., Zhang, L., Khurshid, S., & Perry, D. E. (2015). An information retrieval approach for regression test prioritization based on program changes. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, vol 1. pp. 268–279.

Shepperd, M., Ajienka, N., & Counsell, S. (2018). The role and value of replication in empirical software engineering results. *Information and Software Technology, 99*, 120–132.

Shull, F. J., Carver, J. C., Vegas, S., & Juristo, N. (2008). The role of replications in empirical software engineering. *Empirical Software Engineering, 13*(2), 211–218.

Singh, Y., Kaur, A., Suri, B., & Singhal, S. (2012). Systematic literature review on regression test prioritization techniques. *Informatica (Slovenia), 36*(4), 379–408.

Williamson, D. F., Parker, R. A., & Kendrick, J. S. (1989). The box plot: A simple visual method to interpret data. *Annals of internal medicine, 110*(11), 916–921.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer Science & Business Media.

Yadla, S., Hayes, J. H., & Dekhtyar, A. (2005). Tracing requirements to defect reports: An application of information retrieval techniques. *Innovations in Systems and Software Engineering, 1*(2), 116–124.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability, 22*(2), 67–120.

Yu, T., Srisa-an, W., & Rothermel, G. (2014). Simrt: An automated framework to support regression testing for data races. In: Proceedings of the 36th International Conference on Software Engineering. pp. 48–59.

Zarrad, A. (2015). A systematic review on regression testing for web-based applications. *JSW, 10*(8), 971–990.

Zhang, L. (2018). Hybrid regression test selection. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE. pp. 199–209.

Nasir Mehmood Minhas is a Postdoctoral researcher at the School of Innovation, Design, and Engineering at Mäalrdalen University. He completed his Ph.D. in software engineering from Blekinge Institute of Technology (BTH), Sweden. He has been working on Regression Testing for Embedded Systems in close cooperation with the industry. Nasir has been working in academia since 2000 and has served various institutions. His research interests are software testing, requirement engineering, and software process.

Mohsin Irshad is a software engineer at Ericsson, Sweden, and he received his Ph.D. from Blekinge Institute of Technology (BTH), Sweden, in 2021. Mohsin has a proven track record in the software industry with 10+ years of experience working with different telecommunication vendors. His research interests are in software development, software testing, machine learning, and evidence-based Software Engineering.

Kai Petersen is a professor of software engineering at Blekinge Institute of Technology (BTH), Sweden, and University of Applied Sciences Flensburg, Germany. He received his Ph.D. from BTH in 2010. His research interests are Agile Software Development, Software Testing, Evidence-Based Software Engineering, and Software Measurement. His research has been conducted in close collaboration with companies and with an empirical focus.

Jürgen Börstler is a professor of software engineering and Deputy Head of the Department of Software Engineering at Blekinge Institute of Technology (BTH), Sweden. He received his Ph.D. from Aachen University of Technology (RWTH), Germany in 1993. His research interests are in software process improvement, software quality and measurement, software readability and comprehension, and computer science education.

Authors and Affiliations

Nasir Mehmood Minhas[1,2] · Mohsin Irshad[3] · Kai Petersen[1,4] · Jürgen Börstler[1]

✉ Nasir Mehmood Minhas
nasir.mehmood.minhas@bth.se; nasir.mehmood.minhas@mdu.se

Mohsin Irshad
mohsin.irshad@ericsson.com

Kai Petersen
kai.petersen@bth.se

Jürgen Börstler
jurgen.borstler@bth.se

1 Blekinge Institute of Technology, Karlskrona, Sweden

2 Mäalrdalen University, Västerås, Sweden

3 Ericsson Sweden AB, Karlskrona, Sweden

4 University of Applied Sciences Flensburg, Flensburg, Germany